CSE 132A: Database System Principles
Summer Session I, 2013

Project
(due by Tuesday, July 30, 2pm)

Assume that you want to build a social network for music bands, where music fans can specify the music bands they like. To this end, you have come up with the following relational database schema (which has four relations; each relation name is shown in bold and is followed by the list of attributes of that relation together with the attribute types):

fan(fanID: *integer*, fanName: *string*)
band(bandID: *integer*, bandName: *string*, genre: *string*, year: *integer*)
likes(fanID: *integer*, bandID: *integer*)
bandMember(memID: *integer*, memName: *string*, bandID: *integer*, nationality: *string*)

Primary Keys. The underlined attributes form the primary key of the corresponding relation. For instance, fanID is the primary key of relation **fan**. In the case of relation **likes** both its attributes form *together* its primary key.

Foreign Keys. The schema also contains the following foreign keys (where → means 'references'):
(**likes**.fanID → **fan**.fanID), (**likes**.bandID → **band**.bandID), (**bandMember**.bandID → **band**.bandID)

Other Constraints. The schema also contains a constraint specifying that **band**.year has to be at least 1900 and that **bandMember**.nationality cannot be null.

Schema Semantics. The following explains the semantics of the above schema:
- A tuple (1, 'Joe') in relation **fan** specifies the existence of a fan whose ID is 1 and whose name is 'Joe'.
- A tuple (3, 'Muse', 'rock', 1994) in relation **band** specifies the existence of a band whose ID is 3, its name is 'Muse', its genre 'rock' and its foundation year 1994.
- A tuple (1, 3) in relation **likes** specifies that the fan with fanID = 1 likes the band with bandID = 3.
- A tuple (2, 'Matthew Bellamy', 3, 'British') specifies the existence of a band member whose ID is 2, who is called 'Matthew Bellamy', is a member of the band with bandID = 3 and is of British nationality.

Questions: Do the following in PostgreSQL. It is important to carefully read the project definition and write SQL statements that **cover all corner cases**. As the grading will be for a large part automated (i.e., we will be using scripts to check your queries), it is important to **use the exact names mentioned in the exercise** (for relations, attributes, views etc.).

(1) Write SQL commands that create a database with the above schema, including all the constraints (primary key, foreign key and other constraints). For all string attributes, use the type VARCHAR(100).

(2) Write the following queries in SQL. For each query create a virtual view with the exact name mentioned at the beginning of each question.

   (a) *View bandFans:* List pairs of fans and bands they like. The output schema should be: (fanID, fanName, bandID, bandName).

   (b) *View fanLikeCount:* List for each fan (identified by his ID and name) the number of bands that he likes. If a fan does not like any band, this count should be zero. The output schema should be (fanID, fanName, bandNum).

   (c) *View hateRockBands:* List the fans (identified by their ID and name) that do not like any rock band. The output schema should be (fanID, fanName).

(d) List the genres for which we have the lowest number of bands. The output schema should be (genre) and the output should not contain duplicates. Provide two queries:

(i) *View rareGenresWithMin:* one using the MIN aggregate function

(ii) *View rareGenresWithoutMin:* another without using MIN

(e) List the fans (identified by their ID and name) that like every rock band. The output schema should be (fanID, fanName). Provide three SQL queries, using nested sub-queries in different ways:

i. *View fansAllRockWithNotIn:* with NOT IN tests

ii. *View fansAllRockWithNotExists:* with NOT EXISTS tests

iii. *View fansAllRockWithCount:* with COUNT aggregate functions

(f) *View bandLowerThanAvgFans:* List the bands (identified by their ID and name) that have a strictly lower number of fans than the average number of fans for all bands of the same genre. The output schema should be (bandID, bandName).

(g) *View sameBands:* List pairs of fans (identified by their ID and name) who like the **exact** same bands (i.e., all bands liked by one are also liked by the other and vice versa). Two fans that do not like any band count as liking the exact same bands and therefore should be included in the query answer. The output schema should be (fanID1, fanName1, fanID2, fanName2). The answer should contain only one tuple for each pair of fans that like the exact same bands. For instance, if fans (f1, fN1) and (f2, fN2) like the exact same bands, the answer should include only one of the tuples (f1, fN1, f2, fN2) and (f2, fN2, f1, fN1); not both.

(3) Formulate the following updates in SQL. You may use a sequence of update commands but you are not allowed to change the schema of the database.

(a) Change the nationality of all band members that have a 'british' nationality to 'us' and the nationality of all band members that have *initially* a 'us' nationality to 'british'.

(b) Delete all 'rock' bands, as well as the fans that like them (this has to be done carefully to avoid violation of the referential integrity constraint from **likes** to **band**).

What to turn in: You should turn in three '.sql' files that can be executed in PostgreSQL; one for each exercise (1)-(3). Use the following naming convention (where ID is you student id):

- *File ex1-ID.sql:* Contains solutions to exercise (1).
- *File ex2-ID.sql:* Contains solutions to exercise (2).
- *File ex3-ID.sql:* Contains solutions to exercise (3).

For example, if your student ID is A01234567, then you should submit three files, named 'ex1-A01234567.sql', 'ex2-A01234567.sql' and 'ex3-A01234567.sql'. Each file should include all SQL statements for the corresponding exercise. Make sure to finish each statement with a semicolon (';'), so that each file can be executed in its entirety if given as input to PostgreSQL. Also include as the first line of each file, a line with your name and student ID commented out (you can write a comment as /* comment */).

How to turn it in: You should e-mail the three files to **both** the **instructor** (ikatsis@cs.ucsd.edu) and the **TA** (vpapavas@cs.ucsd.edu) with title: **"CSE132A: Project - Name: xxxx - ID: yyyy"**, where xxxx is your full name and yyyy your student ID number. When sending the e-mail please make sure that: (a) **you have attached all three files with the correct names** and **(b) you have included your student ID and name at the beginning of each file as a comment.** The deadline is **Tuesday, July 30th, 2pm**. No late submissions will be accepted.